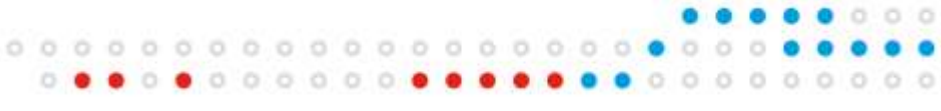
beenen
industrial automation

PLCnext Task Template

C++

Leon Kristiaan



Inhoudsopgave

1 Inleiding

In de PLCnext omgeving kunnen meerdere programma's tegelijk real-time draaien, deze programma's draaien binnen Tasks die worden beheerd door de ESM (Execution and Synchronization Manager).
In de PLCnext omgeving kunnen deze programma's in verschillende talen worden geschreven, namelijk IEC 61131-3, MATLAB Simulink en C++. Dit document bevat alleen informatie over hoe het bijgeleverde templateproject voor C++ werkt.

In dit document zal een poging worden gedaan om alle informatie zo algemeen mogelijk aan te bieden zodat deze in elke development environment kan worden toegepast. Het templateproject heeft de volgende harde eisen:

- **CMake**
- **Make**
- **Phoenix Contact PLCnext SDK**

Dit programma heeft versies voor alle grote Operating Systems. Dit project is echter alleen voor een Unix omgeving ingericht, alhoewel het niet uit maakt of dat *Linux* is of het *Windows Subsystem for Linux*.
Het project kan worden gebruikt voor andere omgevingen, maar dan moeten er waarschijnlijk een aantal aanpassingen worden gedaan.

2 Bestandsstructuur

De bestandsstructuur van het template ziet er als volgt uit:

- build
- cmake
  - template
    - *acf.config.in*
    - *esm.config.in*
    - *gds.config.in*
    - *meta.config.in*
    - *libmeta.in*
    - *compmeta.in*
    - *progmeta.in*
    - *opcua.config.in*
  - *CMakeGen.cmake*
  - *ConfMacro.cmake*
  - *ErrorMacro.cmake*
  - *GenMacro.cmake*
- src
  - Programs
    - *TemplateProgram.hpp*
    - *TemplateProgram.cpp*
  - ProgramProviders
    - *TemplateProgramProvider.hpp*
    - *TemplateProgramProvider.cpp*
  - Components
    - *TemplateComponent.hpp*
    - *TemplateComponent.cpp*
  - *TemplateLibrary.hpp*
  - *TemplateLibrary.cpp*
- *CMakeLists.txt*
- *ProjectConfiguration.cmake*
- *CmakePLCnext.sh*

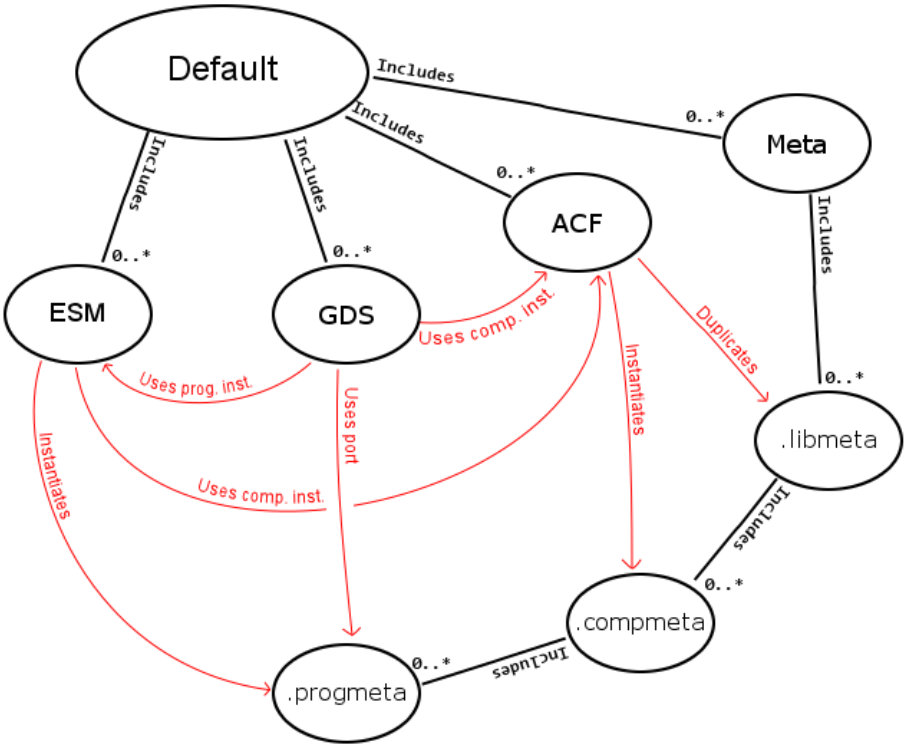
2.1 *build*

De build folder is een lege folder die kan worden gebruikt door CMake om alle build gerelateerde bestanden in aan te maken. Deze techniek wordt ook wel een “out-of-source” build genoemd en heeft als doel het scheiden van source code en compilatie bestanden.

2.2 *cmake*

Deze folder bevat alle bestanden die nodig zijn om configuratie bestanden te bouwen en alle bestanden met hulp functionaliteiten voor CMake. Alleen de templatebestanden hieruit worden behandeld in het document, de functionaliteiten zijn voornamelijk intern en zijn in de bestanden voorzien van commentaar.

Hieronder is een schema opgenomen waarin de relaties staan tussen de verschillende configuratie bestanden:



acf.config.in

Deze configuratie bestanden bevatten verwijzingen naar de *shared objects* van de libraries in het project en ze specificeren instanties van de componenten in de libraries. Dit bestand ziet er ongeveer als volgt uit wanneer ingevuld:

```
<?xml version="1.0" encoding="utf-8"?>
<AcfConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" schemaVersion="1.3"
xmlns="http://www.phoenixcontact.com/schema/acfconfig">
    <Libraries>
        <Library name="TemplateLibrary" binaryPath=
        "$ARP_PROJECTS_DIR$/CPP/Libs/TemplateLibrary/libTemplateLibrary.so"/>
    </Libraries>
    <Components>
        <Component name="TemplateLibrary.TemplateComponentInstance"
        type="TemplateComponent" library="TemplateLibrary">
            <Settings path=""/>
            <Config path=""/>
        </Component>
    </Components>
</AcfConfigurationDocument>
```

De informatie in dit bestand over waar de library staat komt overeen met wat er in de .libmeta metadata configuratie te vinden is. Het type van de component komt uit de .compmeta metadata configuratie.

Gedetailleerde informatie over het format van deze bestanden is hier te vinden:

https://www.plcnext-community.net/index.php?option=com_content&view=article&id=136&catid=35&Itemid=253&lang=en

esm.config.in

Deze configuratie bestanden bevatten informatie over welke taken er moeten worden aangemaakt door de ESM en wat voor taken het zijn. Ze leggen ze een link tussen de taken en de programma's die daarin draaien. Ze specificeren verder ook instanties van de programma's. Dit bestand ziet er ongeveer als volgt uit wanneer ingevuld:

```
<?xml version="1.0" encoding="utf-8"?>
<EsmConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" schemaVersion="1.0"
xmlns="http://www.phoenixcontact.com/schema/esmconfig">
    <Tasks>
        <CyclicTask name="Cyclic1000" stackSize="0" priority="1"
        cycleTime="1000000000" watchdogTime="1000000000"
        executionTimeThreshold="0" />
    </Tasks>
    <EsmTaskRelations>
        <EsmTaskRelation esmName="ESM1" taskName="Cyclic1000" />
    </EsmTaskRelations>
    <Programs>
        <Program name="TemplateProgramInstance" programType="TemplateProgram"
        componentName="TemplateLibrary.TemplateComponentInstance" />
    </Programs>
    <TaskProgramRelations>
        <TaskProgramRelation taskName="Cyclic1000" programName=
        "TemplateLibrary.TemplateComponentInstance/TemplateProgramInstance"
        order="0" />
    </TaskProgramRelations>
</EsmConfigurationDocument>
```

De informatie over het programma type komt overeen met wat er in de .progmeta te vinden is, de component instantie waar hier gebruik van wordt gemaakt is in de ACF-configuratie te vinden.

Gedetailleerde informatie over het format van deze bestanden is hier te vinden:
https://www.plcnext-community.net/index.php?option=com_content&view=article&id=43&catid=35&Itemid=253&lang=en

gds.config.in
Deze configuratie bestanden bevatten informatie over welke connecties er gelegd moeten worden tussen de in/out porten van dit project en andere in/out porten die beschikbaar zijn in de GDS. Deze connecties hoeven maar door één programma te worden gelegd. Dit bestand ziet er ongeveer als volgt uit wanneer ingevuld:

```
<?xml version="1.0" encoding="utf-8"?>
<GdsConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" schemaVersion="1.0"
xmlns="http://www.phoenixcontact.com/schema/gdsconfig">
    <Connectors>
        <Connector startPort=
        "TemplateLibrary.TemplateComponentInstance/TemplateProgramInstance:OP_Byte"
        endPort="Arp.Io.FbIo.PnC/96:Output Byte"/>
    </Connectors>
</GdsConfigurationDocument>
```

De library naam en de component instantie naam zijn te vinden in de ACF-configuratie. De programma instantie naam is te vinden in de ESM configuratie en de port naam is te vinden in de .progmeta metadata configuratie.
De naam van een port buiten het programma is in de configuratie van een ander programma te vinden, of in PC Worx Engineer als het om een IEC programma gaat.

Gedetailleerde informatie over het format van deze bestanden is hier te vinden:
https://www.plcnext-community.net/index.php?option=com_content&view=article&id=68&catid=35&Itemid=253&lang=en

meta.config.in
Deze configuratie bestanden bevatten verwijzingen naar de metadata informatie van het project. Dit is een redelijk simpel bestand en het ziet er als volgt uit wanneer ingevuld:

```
<?xml version="1.0" encoding="utf-8"?>
<MetaConfigurationDocument>
    <MetaIncludes>
        <MetaInclude path="../../Libs/TemplateLibrary" />
    </MetaIncludes>
</MetaConfigurationDocument>
```

libmeta.in

Deze metadata bestanden bevatten informatie over de libraries zelf en verwijzingen naar waar informatie over de componenten uit die library te vinden is. Dit bestand ziet er ongeveer als volgt uit wanneer ingevuld:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MetaConfigurationDocument xmlns="http://www.phoenixcontact.com/schema/metaconfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" schemaVersion="1.0">
    <Library name="TemplateLibrary" applicationDomain="CPLUSPLUS">
        <File path="libTemplateLibrary.so" checksum="" />
        <ComponentIncludes>
            <Include path="TemplateLibrary_C/TemplateLibrary_C.compmeta"/>
        </ComponentIncludes>
    </Library>
</MetaConfigurationDocument>
```

compmeta.in

Deze metadata bestanden bevatten informatie over de componenten van een library en verwijzingen naar waar informatie over de programma's uit die componenten te vinden is. Dit bestand ziet er ongeveer als volgt uit wanneer ingevuld:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MetaConfigurationDocument xmlns="http://www.phoenixcontact.com/schema/metaconfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" schemaVersion="1.0">
    <Component name="TemplateComponent">
        <ProgramIncludes>
            <Include path=" TemplateLibrary_P/TemplateLibrary_P.progmeta" />
        </ProgramIncludes>
    </Component>
</MetaConfigurationDocument>
```

progmeta.in

Deze metadata bestanden bevatten informatie over de programma's van een component en welke in/out porten zijn gedefinieerd. Dit bestand ziet er ongeveer als volgt uit wanneer ingevuld:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MetaConfigurationDocument schemaVersion="1.0"
xmlns="https://www.phoenixcontact.com/schema/metaconfig">
    <Program name="TemplateProgram">
        <Ports>
            <Port kind="Output" name="OP_Byte" type="uint8" multiplicity="1"/>
        </Ports>
    </Program>
</MetaConfigurationDocument>
```

opcua.config.in

Deze configuratie bevat informatie over wat de OPC UA server mag zien van het programma en hoe het dit moet tonen. Dit bestand ziet er ongeveer als volgt uit:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<OpcUAConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.phoenixcontact.com/schema/opcuaconfig"
xmlns="http://www.phoenixcontact.com/schema/opcuaconfig">
    <NodeName>TemplateLibrary</NodeName>
    <ServerCertificate>
        <SelfSigned />
    </ServerCertificate>
    <GdsPortsToProvide>
        <All />
    </GdsPortsToProvide>
</OpcUAConfigurationDocument>
```

Voor zover als ik het kan bepalen besluit de **NodeName** tag hoe dit project moet worden getoond in de OPC UA server. Verder bepaald de **GdsPortsToProvide** lijst informatie over welke ports er getoond mogen worden, maar ik heb geen informatie kunnen vinden over het format hiervoor. De enige bekende waarden voor deze instelling zijn **All** en **None**.

2.3 *src*

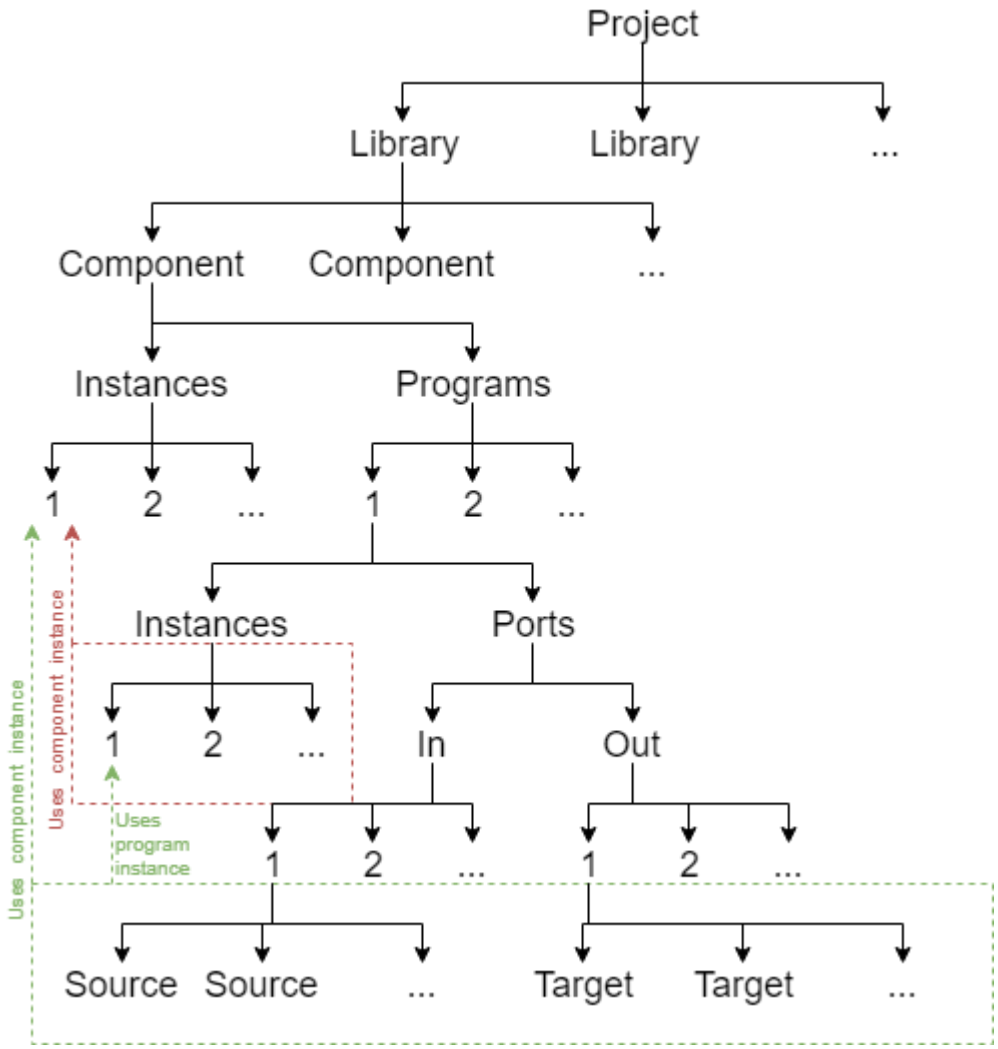
De src folder bevat de daadwerkelijke code van het project, hierin is de code opgenomen voor de library, de componenten en de programma's.
Voor gedetailleerde informatie hierover verwijs ik naar de voorbeeld projecten van Phoenix Contact en ook naar hun eigen uitleg hierover die hier te vinden is:
https://www.plcnext-community.net/index.php?option=com_content&view=article&id=41&catid=45&Itemid=263&lang=en

3 Configuratie

Elk project heeft zijn eigen configuratie nodig, om dit voor de gebruiker van dit template makkelijker te maken zijn er een aantal configuratie functies gemaakt in CMake. De project configuratie kan in zijn geheel worden gevonden in 1 bestand:

- **ProjectConfiguration.cmake**

Hieronder is een schema opgenomen wat deze configuratie ongeveer illustreert:



Enige toelichting voor dit schema is nodig:
Een project is de omgeving waar je in werkt, dit bevat niet alleen broncode maar ook configuratie bestanden. In een project kunnen meerdere Libraries worden gemaakt met elk hun eigen doel.
Een Library kan meerdere componenten bevatten die een soort los ecosysteem vormen waarin weer verschillende programma's kunnen worden gemaakt.
De Programma's bevatten de code die individuele functionaliteiten realiseren van een Library en zijn ook de plek waar GDS poorten worden gedefinieerd.
Om elk van de lagen in te vullen zijn er functies aanwezig in de CMake omgeving, deze worden op de komende pagina's één voor één toegelicht.

3.1 *PLCnext*

Voor standalone projecten zijn er een aantal configuratie bestanden die aangepast moeten worden om het systeem te vertellen waar het project te vinden is. De bestanden die hiervoor moeten worden aangepast zijn de volgende:

- `/opt/plcnext/projects/Default/Plc/Esm/Default.esm.config`
- `/opt/plcnext/projects/Default/Plc/Gds/Default.gds.config`
- `/opt/plcnext/projects/Default/Plc/Plm/Plm.acf.config`
- `/opt/plcnext/projects/Default/Plc/Meta/Default.meta.config`

Default.esm.config

Dit bestand verwijst door naar de locatie van de esm configuratie bestanden van projecten.
In dit bestand moeten er aanpassingen worden gemaakt in de volgende regels:

```
<Includes>
    <Include path="$ARP_PROJECTS_DIR$/PCWE/Plc/Esm/*.esm.config" />
    <Include path="Globals.esm.config" />
    <Include path="ServiceTask.esm.config" />
</Includes>
```

Aan deze regels moet een nieuwe Include worden toegevoegd, bijvoorbeeld:

```
    <Include path="$ARP_PROJECTS_DIR$/CPP/Plc/Esm/*.esm.config" />
```

Default.gds.config

Dit bestand verwijst door naar de locatie van de gds configuratie bestanden van projecten.
In dit bestand moeten er aanpassingen worden gemaakt in de volgende regels:

```
<Includes>
    <Include path="$ARP_PROJECTS_DIR$/PCWE/Plc/Gds/*.gds.config" />
</Includes>
```

Aan deze regels moet een nieuwe Include worden toegevoegd, bijvoorbeeld:

```
    <Include path="$ARP_PROJECTS_DIR$/CPP/Plc/Gds/*.gds.config" />
```

Plm.acf.config
Dit bestand verwijst door naar de locatie van de acf configuratie bestanden van projecten.
In dit bestand moeten er aanpassingen worden gemaakt in de volgende regels:

```
<Includes>
    <Include path="$ARP_PROJECTS_DIR$/PCWE/Plc/Plm/*.acf.config" />
</Includes>
```

Aan deze regels moet een nieuwe Include worden toegevoegd, bijvoorbeeld:

```
    <Include path="$ARP_PROJECTS_DIR$/CPP/Plc/Plm/*.acf.config" />
```

Default.meta.config
Dit bestand verwijst door naar de locatie van de meta configuratie bestanden van projecten.
In dit bestand moeten er aanpassingen worden gemaakt in de volgende regels:

```
<Includes>
    <Include path="$ARP_PROJECTS_DIR$/PCWE/Plc/Meta/*.meta.config" />
</Includes>
```

Aan deze regels moet een nieuwe Include worden toegevoegd, bijvoorbeeld:

```
    <Include path="$ARP_PROJECTS_DIR$/CPP/Plc/Meta/*.meta.config" />
```

3.2 *Project*

Het project in de context van configuratie verwijst naar de naam van de folder waarin het project zich op de PLCnext zal bevinden. Voor IEC-applicaties vanuit PC Worx Engineer zou dit “PCWE” zijn, voor het templateproject is dit “CPP”.
Ook heeft het project een pad nodig om te weten waar de SDK van PLCnext te vinden is.

`plcnext_library_builder ( exe_pad )`

Deze functie vult het pad in waar de PC Worx Library Builder op het systeem staat.

Parameters

`exe_pad`

Een absoluut pad naar de locatie van de Library Builder executable.

Voorbeeld(en)

```
plcnext_library_builder("/opt/EngineeringLibraryBuilder.exe")
```

`plcnext_root_dir ( sdk_pad )`

Deze functie vult het pad in waar de PLCnext SDK staat op het systeem zodat CMake hier de header files vandaan kan halen.

Parameters

`sdk_pad`

Een absoluut pad naar de hoofdfolder van de PLCnext SDK

Voorbeeld(en)

```
plcnext_root_dir("/opt/pxc/2.2.1/sysroots/cortexa9t2hf-neon-pxc-linux-gnueabi")
```

`plcnext_project_name ( project_naam )`

Deze functie vult de naam van het project in. De naam wordt gebruikt om te bepalen waar het project op de PLCnext staat.

Parameters

`project_naam`

De naam van de folder waar het project in staat wanneer op de PLCnext.

Voorbeeld(en)

```
plcnext_project_name("CPP")
```

`plcnext_add_include ( include_pad )`

Deze functie voegt een pad toe aan de lijst van include paden. Dit pad verwijst bijvoorbeeld naar externe library headers die nodig zijn om het project te bouwen.

Parameters

`include_pad`

Het pad naar de headers van bijv. een externe library.

Voorbeeld(en)

```
plcnext_add_include("/usr/include")
```

3.3 *Libraries*

`plcnext_add_library ( lib_naam )`

Deze functie voegt een library toe aan de lijst van libraries die voor dit project moeten worden gebouwd.

Parameters

`lib_naam`

De naam van de library zoals in de source-code te vinden is.

Voorbeeld(en)

```
plcnext_add_library("TemplateLib")
```

3.4 *Components*

`plcnext_add_component ( lib_naam comp_naam )`

Deze functie voegt een component toe aan een library.

Parameters

`lib_naam`

De naam van de library zoals gedefinieerd met *`plcnext_add_library`*.

`comp_naam`

De naam van de component zoals in de source-code te vinden is.

Voorbeeld(en)

```
plcnext_add_component("TemplateLib" "TemplateComp")
```

plcnext_add_component_instance (lib_naam comp_naam inst_naam)

Deze functie definieert een instantie van een component.

Parameters

`lib_naam`

De naam van de library zoals gedefinieerd met *plcnext_add_library*.

`comp_naam`

De naam van een component zoals gedefinieerd met *plcnext_add_component*.

`inst_naam`

De naam van de instantie.

Voorbeeld(en)

```
plcnext_add_component_instance("TemplateLib" "TemplateComp" "TemplateCompInst")
```

3.5 *Programs*

plcnext_add_program (lib_naam comp_naam prog_naam)

Deze functie voegt een programma toe aan een component.

Parameters

`lib_naam`

De naam van de library zoals gedefinieerd met *plcnext_add_library*.

`comp_naam`

De naam van een component zoals gedefinieerd met *plcnext_add_component*.

`prog_naam`

De naam van een programma zoals in de source-code te vinden is.

Voorbeeld(en)

```
plcnext_add_program("TemplateLib" "TemplateComp" "TemplateProg")
```

plcnext_add_program_instance
(lib_naam comp_naam comp_inst prog_naam prog_inst)

Deze functie definieert een instantie van een programma.

Parameters

`lib_naam`

De naam van de library zoals gedefinieerd met *plcnext_add_library*.

`comp_naam`

De naam van een component zoals gedefinieerd met *plcnext_add_component*.

`comp_inst`

De naam van een instantie van het *comp_naam* component zoals gedefinieerd met *plcnext_add_component_instance*.

`prog_naam`

De naam van een programma zoals gedefinieerd met *plcnext_add_program*.

`prog_inst`

De naam van de instantie.

Voorbeeld(en)

```
plcnext_add_program_instance("TemplateLib" "TemplateComp" "TemplateCompInst"
"TemplateProg" "TemplateProgInst")
```

3.6 *Ports*

plcnext_add_program_port
(lib_naam comp_naam prog_naam port_naam port_type port_aant port_richt)

Deze functie voegt een port toe aan een programma.

Parameters

`lib_naam`
: De naam van de library zoals gedefinieerd met *`plcnext_add_library`*.

`comp_naam`
: De naam van een component zoals gedefinieerd met *`plcnext_add_component`*.

`prog_naam`
: De naam van een programma zoals gedefinieerd met *`plcnext_add_program`*.

`port_naam`
: De naam van de port.

`port_type`
: Het type van de data in de port. (Zie ondersteunde datatypes van Phoenix voor details.)

`port_aant`
: De hoeveelheid keer dat het datatype in de port voorkomt. (De lengte van een array.)

`port_richt`
: De richting van de port, dit kan *`Output`* of *`Input`* zijn.

Voorbeeld(en)

```
plcnext_add_program_port("TemplateLib" "TemplateComp" "TemplateProg" "OP_Bit" "bit" "1" "Output")
```

```
plcnext_add_program_port("TemplateLib" "TemplateComp" "TemplateProg" "OP_Byte" "uint8" "1" "Output")
```

```
plcnext_add_program_port("TemplateLib" "TemplateComp" "TemplateProg" "IP_Bit" "bit" "1" "Output")
```

plcnext_add_program_instance_port_out
(lib_naam comp_naam prog_naam prog_inst port_naam doel)

Deze functie verbindt een *Output* met een *Input* port van een ander programma.

Parameters

`lib_naam`

De naam van de library zoals gedefinieerd met *`plcnext_add_library`*.

`comp_naam`

De naam van een component zoals gedefinieerd met *`plcnext_add_component`*.

`prog_naam`

De naam van een programma zoals gedefinieerd met *`plcnext_add_program`*.

`prog_inst`

De naam van een programma instantie zoals gedefinieerd met *`plcnext_add_program_instance`*.

`port_naam`

De naam van een *`Ouput`* port zoals gedefinieerd met *`plcnext_add_program_port`*.

`doel`

Een verwijzing naar een *`Input`* port van een ander programma.

Voorbeeld(en)

```
plcnext_add_program_instance_port_out("TemplateLib" "TemplateComp" "TemplateProg"
"TemplateProgInst" "OP_Bit" "TemplateLib.TemplateCompInst/TemplateProgInst:IP_Bit")
```

```
plcnext_add_program_instance_port_out("TemplateLib" "TemplateComp" "TemplateProg"
"TemplateProgInst" "OP_Byte" "Arp.Plc.Eclr/MainInstance:Input Byte")
```

```
plcnext_add_program_instance_port_out("TemplateLib" "TemplateComp" "TemplateProg"
"TemplateProgInst" "OP_Byte" "Arp.Io.Fbio.Pnc/96:Output Byte")
```

`plcnext_add_program_instance_port_in`
`( lib_naam comp_naam prog_naam prog_inst port_naam bron )`

Deze functie verbindt een *Output* port van een ander programma met een *Input* port.

Parameters

`lib_naam`

De naam van de library zoals gedefinieerd met *`plcnext_add_library`*.

`comp_naam`

De naam van een component zoals gedefinieerd met *`plcnext_add_component`*.

`prog_naam`

De naam van een programma zoals gedefinieerd met *`plcnext_add_program`*.

`prog_inst`

De naam van een programma instantie zoals gedefinieerd met *`plcnext_add_program_instance`*.

`port_naam`

De naam van een *`Input`* port zoals gedefinieerd met *`plcnext_add_program_port`*.

`bron`

Een verwijzing naar een *`Output`* port van een ander programma.

Voorbeeld(en)

```
plcnext_add_program_instance_port_in("TemplateLib" "TemplateComp" "TemplateProg"
"TemplateProgInst" "IP_Bit" "TemplateLib.TemplateCompInst/TemplateProgInst:OP_Bit")
```

3.7 *Tasks*

`plcnext_add_task`
`( task_naam task_stack task_prio task_cycle task_watch task_thres )`

Deze functie voegt een nieuwe cyclische taak toe aan het project.

Parameters

`task_naam`

De naam van de taak

`task_stack`

`task_prio`

De prioriteit van de taak, deze waarde gaat van 0 tot 31. Hoe lager de waarde hoe meer prioriteit de taak heeft.

`task_cycle`

De tijd tussen uitvoeringen van de taak in nanoseconden.

`task_watch`

De tijd in nanoseconden voordat er wordt gekeken of de taak nog draait.

`task_thres`

Voorbeeld(en)

```
plcnext_add_task("Cyclic1000" "0" "1" "1000000000" "1000000000" "0")
```

```
plcnext_add_task("Cyclic100" "0" "1" "100000000" "100000000" "0")
```

`plcnext_assign_task ( task_naam esm_naam )`

Deze functie wijst een taak naar de ESM waar hij op moet draaien.

Parameters

`task_naam`

De naam van een taak gedefinieerd met *`plcnext_add_task`*.

`esm_naam`

De naam van een ESM. (ESM1, ESM2, …)

Voorbeeld(en)

```
plcnext_assign_task("Cyclic100" "ESM1")
```

`plcnext_assign_program_instance ( lib_naam comp_inst prog_inst task_naam )`

Deze functie wijst een programma instantie naar de taak waar hij in draait.

Parameters

`lib_naam`

De naam van de library zoals gedefinieerd met *`plcnext_add_library`*.

`comp_inst`

De naam van een instantie van het *`comp_naam`* component zoals gedefinieerd met *`plcnext_add_component_instance`*.

`prog_inst`

De naam van een programma instantie zoals gedefinieerd met *`plcnext_add_program_instance`*.

`task_naam`

De naam van een taak gedefinieerd met *`plcnext_add_task`*.

Voorbeeld(en)

```
plcnext_assign_program_instance("TemplateLib" "TemplateCompInst" "TemplateProgInst"
"Cyclic100")
```

3.8 *Voorbeelden*

Hieronder zijn een aantal voorbeeld configuraties opgenomen.

Hieronder staat een configuratie bestand van het template project (*CPP*), waar 1 Library in aanwezig is, met 1 Component en 1 Program. Het programma heeft 2 output poorten en die zijn verbonden met de profinet controller en een programma uit een PCWE project.
Ook wordt er een ESM task gespecificeerd zodat het programma standalone kan draaien.

```
plcnext_root_dir("/opt/pxc/2.2.1/sysroots/cortexa9t2hf-neon-pxc-linux-gnueabi")
plcnext_project_name("CPP")

plcnext_add_library("TemplateLibrary")
plcnext_add_component("TemplateLibrary" "TemplateComponent")
plcnext_add_component_instance("TemplateLibrary" "TemplateComponent"
                               "TemplateComponentInstance")
plcnext_add_program("TemplateLibrary" "TemplateComponent" "TemplateProgram")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "OP_Bit" "bit" "1" "Output")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "OP_Byte" "uint8" "1" "Output")
plcnext_add_program_instance("TemplateLibrary" "TemplateComponent"
                             "TemplateComponentInstance" "TemplateProgram"
                             "TemplateProgramInstance")
plcnext_add_program_instance_port_out("TemplateLibrary" "TemplateComponent"
                    "TemplateProgram" "TemplateProgramInstance" "OP_Bit"
                    "Arp.Plc.Eclr/MainInstance:Input Bit")
plcnext_add_program_instance_port_out("TemplateLibrary" "TemplateComponent"
                    "TemplateProgram" "TemplateProgramInstance" "OP_Byte"
                    "Arp.Io.FbIo.PnC/96:Output Byte")

plcnext_add_task("Cyclic1000" "0" "1" "1000000000" "1000000000" "0")
plcnext_assign_task("Cyclic1000" "ESM1")
plcnext_assign_program_instance("TemplateLibrary" "TemplateComponentInstance"
                                "TemplateProgramInstance" "Cyclic1000")
```

Hieronder staat een configuratie voor een variatie van het template project, hier heeft het programma 1 input poort en zijn er 3 instanties van in plaats van 1. Maar 1 van de programma instanties heeft zijn input poort gekoppeld met een PCWE project.
Dit is ook een project wat standalone kan draaien.

```
plcnext_root_dir("/opt/pxc/2.2.1/sysroots/cortexa9t2hf-neon-pxc-linux-gnueabi")
plcnext_project_name("CPP")

plcnext_add_library("TemplateLibrary")
plcnext_add_component("TemplateLibrary" "TemplateComponent")
plcnext_add_component_instance("TemplateLibrary" "TemplateComponent"
                               "TemplateComponentInstance")
plcnext_add_program("TemplateLibrary" "TemplateComponent" "TemplateProgram")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "IP_Bit" "bit" "1" "Input")
plcnext_add_program_instance("TemplateLibrary" "TemplateComponent"
                             "TemplateComponentInstance" "TemplateProgram"
                             "TemplateProgramInstance1")
plcnext_add_program_instance("TemplateLibrary" "TemplateComponent"
                             "TemplateComponentInstance" "TemplateProgram"
                             "TemplateProgramInstance2")
plcnext_add_program_instance("TemplateLibrary" "TemplateComponent"
                             "TemplateComponentInstance" "TemplateProgram"
                             "TemplateProgramInstance3")
plcnext_add_program_instance_port_in("TemplateLibrary" "TemplateComponent"
                    "TemplateProgram" "TemplateProgramInstance3" "IP_Byte"
                    "Arp.Plc.Eclr/:Input")

plcnext_add_task("Cyclic1000" "0" "1" "1000000000" "1000000000" "0")
plcnext_assign_task("Cyclic1000" "ESM1")
plcnext_assign_program_instance("TemplateLibrary" "TemplateComponentInstance"
                                "TemplateProgramInstance1" "Cyclic1000")
plcnext_assign_program_instance("TemplateLibrary" "TemplateComponentInstance"
                                "TemplateProgramInstance2" "Cyclic1000")
plcnext_assign_program_instance("TemplateLibrary" "TemplateComponentInstance"
                                "TemplateProgramInstance3" "Cyclic1000")
```

Hieronder is een configuratie opgenomen die specificeert waar de Library Builder te vinden is. Er wordt hier een stuk minder in geconfigureerd en er worden geen instanties gespecificeerd van de Component en Program. Zonder instanties kunnen er ook geen tasks worden gekoppeld en kan dit project niet standalone draaien, het kan alleen als Library worden gebouwd.

```
plcnext_library_builder("/opt/EngineeringLibraryBuilder.exe")
plcnext_root_dir("/opt/pxc/2.2.1/sysroots/cortexa9t2hf-neon-pxc-linux-gnueabi")
plcnext_project_name("CPP")

plcnext_add_library("TemplateLibrary")
plcnext_add_component("TemplateLibrary" "TemplateComponent")
plcnext_add_program("TemplateLibrary" "TemplateComponent" "TemplateProgram")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "IP_Bit" "bit" "1" "Input")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "OP_Bit" "bit" "1" "Output")
```

4 Compilatie

Dit template maakt gebruik van *CMake*, dit is een programma waar online veel informatie over te vinden is maar in het kort is het een programma wat compilatie configuraties kan maken voor veel verschillende build omgevingen. Het template project maakt gebruik van *Make*.
Ook wordt er verwezen naar de locatie van de PLCnext SDK van Phoenix Contact.

De eerste stap die er moet worden genomen voor het compileren van het programma is met CMake de configuratie bestanden voor Make genereren, in het template project is er een shell script aangeleverd wat dit kan doen en wat gelijk de PLCnext SDK verwijzingen opneemt:

```
./CMakePLCnext.sh
```

Intern voert dit script het volgende commando uit:

```
cmake .. -DCMAKE_INSTALL_PREFIX=../install -DCMAKE_BUILD_TYPE=release
```

Dit houdt in dat CMake zijn configuratie uit de bovenliggende folder haalt en dat de installatie van het project in de *install* folder plaatst. Het plaatst de configuratie bestanden die het genereerd in de zelfde folder als waar dit commando wordt uitgevoerd.

Dit script verwijst naar de volgende locatie voor het setup bestand van de PLCnext SDK:

```
/opt/pxc/2.2.1/environment-setup-cortex9t2hf-neon-pxc-linux-gnueabi
```

Dit pad moet worden aangepast in geval de SDK niet op de standaard plek staat. Als deze verwijzing klopt dan kan het bestand worden uitgevoerd en zal het de configuratie bestanden aanmaken voor Make.
Normaal worden de configuratie bestanden voor het compileren in een aparte folder geplaats, een **build** folder.

Als de configuratie bestanden zijn gemaakt kan Make worden uitgevoerd in de zelfde folder als waar CMake is uitgevoerd, hiervoor zijn twee verschillende commando's.

Standalone

Voor een project wat los op de PLCnext draait is er een commando om de hele projectstructuur klaar te maken zodat deze direct kan worden gekopieerd:

```
make install
```

Hierdoor wordt er in de folder die gespecificeerd is met **–DCMAKE_INSTALL_PREFIX** het project geplaatst, klaar voor gebruik.

PC Worx Engineer Library

Voor een project wat als Library wordt gebruikt in PC Worx Engineer is er een ander commando, dit commando kan alleen worden gebruikt als er in de configuratie de locatie van de Library Builder is aangewezen.

```
make _PCWE_Build_Library
```

Hierdoor wordt er in de folder waar dit uit wordt gevoerd, de **build** folder, een .pcwlx bestand gemaakt. Dit bestand kan in PC Worx Engineer worden ingeladen als een Library en zal de Libraries, Componenten en Programma's van het project bevatten.

5 Installatie

Nadat een project is gecompileerd is er een laatste stap nodig om het project in gebruik te nemen, en dat is het installeren van het project op de juiste plek. Deze instructies zijn enorm verschillend voor standalone projecten en PC Worx Engineer Libraries.

Standalone

Voor de standalone projecten zijn er configuratie bestanden in de PLCnext die aangepast moeten worden, deze zijn uitgelegd in hoofdstuk *3.1: PLCnext*.
Voor de locatie van de configuratie bestanden van het project moet het projectnaam in gedachten worden gehouden, dat is namelijk de naam van een folder in de projecten folder van PLCnext.

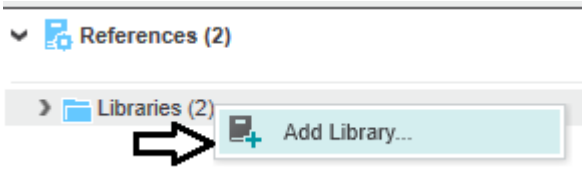
Nadat die configuratie stappen zijn gedaan kan er met een FTP programma de volgende folder aangemaakt worden:

```
/opt/plcnext/projects/<Naam van project zoals gezet met plcnext_project_name()>
```

In die folder kan de gehele inhoud van de installatie folder worden geüpload.

PC Worx Engineer Library

Voor PC Worx Engineer Libraries is het zeer simpel, er is rechts onder een gebied met “References” te vinden, als je rechter muisknop drukt op de “Libraries” item die daar in zit krijg je het onderstaande:



Hierna krijg je een dialoog waarin je kan bladeren naar de locatie van het .pcwlx bestand en deze inladen. Dit maakt nieuwe programma’s beschikbaar in het PC Worx Engineer project die vervolgens daar kunnen worden geconfigureerd.